ANALOG PUBLIC PUF FOR HARDWARE SECURITY

A Thesis
Presented to
The Academic Faculty

by

Kyle Andrew Gassel

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
May 2018

ANALOG PUBLIC PUF FOR HARDWARE SECURITY

Approved by:

Dr. Abhijit Chatterjee, Advisor

School of Electrical and Computer Engineering

Georgia Institute of Technology

Dr. Linda S. Milor

School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. David C. Keezer

School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: May 27, 2018

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Abhijit Chatterjee, who first got me interested in security whose guidance and support has been invaluable at every step in the process. I would also like to thank the members of my lab, particularly Barry Muldrey, whose expertise and creativity helped me out of many ruts, and Sujay Pandey, whose advice and support was always there when I needed it most.

Finally I would like to thank my manager and supervisor, Shawn Kerr, without whom I would have not had the opportunity to attend Georgia Tech. His belief and investment in me made this possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

PUF	Physically Unclonable Function
PPUF	Public Physically Unclonable Function
APPUF	Analog Public Physically Unclonable Function
ESG	Execution-Simulation Gap
ADC	Analog to Digital Converter
DAC	Digital to Analog Converter
LPF	Low Pass Filter
HPF	High Pass Filter
LUT	Look Up Table

SUMMARY

A secure Analog Public PUF (Physically Unclonable Function) would be a significant advancement for computer security. It offers several advantages over digital PUFs, but it has not yet been shown that one could satisfy reasonable demands of repeatability, uniqueness, and resistance to attacks. In this thesis I improve a current analog PUF design, primarily in uniqueness and reliability, and further analyze it to show fitness for use as a PPUF.

CHAPTER 1. INTRODUCTION

1.1 PUFs

A PUF, or Physically Unclonable Function, is a secure hardware device that takes a challenge and produces a unique but recognizable response. These have significant value due to the implication that one would have to physically steal the PUF in order to get access to the secured system it protects. Standard PUFs rely on the fact that the circuit is unknown to the public, and that its function is complex enough that responses cannot be guessed given the inputs. This has some major weaknesses. Some PUFs can be vulnerable to modeling attacks, in which the attacker models the characteristics of a PUF well enough to guess the function [1]. Others rely on a limited set of challenges that could be stored in a look-up table. Finally, since all PUFs rely on the secrecy of the design, this could be compromised without the owner's knowledge.

1.2 PPUFs

Public PUFs are functionally very similar but allow for wider, albeit more nuanced, applications. The internal circuitry of the PUF is modeled before distribution, and the model is released to the public so that anyone can simulate a challenge-response pair. The security of PPUFs rely not on any secrecy, but on the wide discrepancy between response time of a physical PPUF and simulation of its model. This is known as the Execution-Simulation Gap, or ESG. To verify the owner of a PPUF, one can simulate a particular challenge with their model, then ask the PPUF owner to return the correct response for that

challenge within a set period of time. If the response comes fast enough, the owner must have the actual PPUF.

1.2.1 PPUF Fitness Metrics

There are three primary measures that make a good PPUF. They are:

- Uniqueness – Since the properties of each PPUF are determined by process variations in a common design, the output must adequately amplify these differences so that as many unique PUFs as possible can be obtained from a set number of circuits produced.
- Reliability – PUFs must be reliable, primarily over temperature, so that they can be identified properly in a variety of environments.
- Resistance to attacks – PPUFs are relatively resistant to attacks due to their public nature, but there are still a few types of attacks that must be taken into account. The most obvious are simulation attacks, in which the attacker is able to simulate the PPUF fast enough to appear to be one. Attackers can also build hardware or firmware based on the model to closely approximate a specific PPUF. This second type of attack is a major weakness of most private PUFs, as they rely on the secrecy of the model to prevent reverse engineering attacks.
- Entropy – Although the model is public, it still has to be sufficiently complex. Entropy mostly feeds back into uniqueness, which requires small process variations to result in an observable change in the output, and resistance to attack,

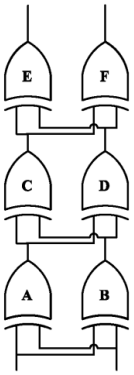
in which changes in either the challenge or the process variation must result in a change in the response that is hard to predict.

CHAPTER 2. STATE OF THE ART

The two main PPUFs that are comparable to the new Analog PPUF design are the Arbiter PUF, a standard in the field, and the relatively new "NanoPPUF" bidirectional PUF.

2.1 Arbiter PUF

Arbiter PUFs build on the very first PUF design. They are based on XOR gates strung together, as seen in a heavily simplified configuration in Figure 1, to produce variable but repeatable glitching at the output [2].



Gate	Input 1 Delay (ps)	Input 2 Delay (ps)
A	9.3	10.1
B	11.2	8.8
C	11.1	9.0
D	7.8	7.1
E	8.6	9.5
F	12.4	9.6

Figure 1 – Simplified Arbiter PUF [3]

The glitches change with process variation and different inputs, making it a good candidate for a PUF. Two of these are placed in parallel and fed into an arbiter, which gives a simple bit output based on which circuit had a higher delay. This removes the need for prohibitively accurate sensing of the output signal timing. Arbiter PUFs make very secure private PUFs because they operate by obfuscating simple building blocks. This makes them very weak PPUFs, however, because once the model is known, as is a requirement for a

PPUF, one can build reverse-engineered hardware or firmware with an insignificantly small ESG [4].

2.2 "Nano" Network PUF

"Nano" PUFs diverge from delay-based sensing by creating a large network of simple connections between ports and nodes [5]. NanoPPUFs use memristors formed at random points on a grid between a set number of bidirectional ports to create circuit blocks, as seen in Figure 2.

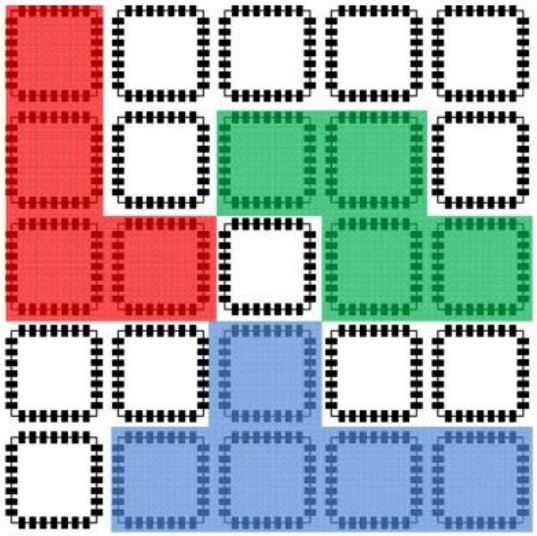


Figure 2 – NanoPPUF array with adjacent cells to be connected highlighted for three possible configurations

These blocks form an array and can be connected in an exponential number of ways to form unique circuits. Since the inputs, configuration, and outputs can be controlled, it would take prohibitively long to simulate the entire NanoPPUF. Only a few outputs are required to verify the identity of the circuit, however, so verification simulations can be run relatively quickly. This design has not been studied in depth to determine whether it

meets the metrics of repeatability, uniqueness, and security. It is likely susceptible to modeling attacks due to the simple nature of the blocks. One paper in particular suggests that parallel matrix multiplication methods could defeat the PUF fairly easily [6].

CHAPTER 3. ANALOG PPUF DESIGN

3.1 Previous Work

The design outlined here is based on work done by Sabyasachi Deyati as part of his dissertation work. In a paper he designs a PUF using a simple push-pull inverter circuit [7]. The push-pull transistors are small in size to increase the effect of process variations. Deyati et al. also added biasing resistors, seen in Figure 3, to make the simulation harder by preventing clipping at the rails.

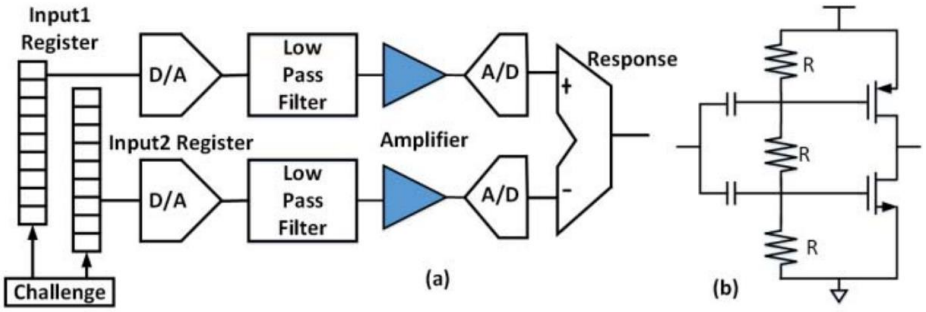


Figure 3 – (a) Analog PUF architecture (b) Push-pull amplifier

Instead of taking a set of inputs and waiting for an output to arrive like traditional delay-based PUFs, the Analog PUF takes a continuous analog challenge as an input to a push-pull inverter and samples the output of the inverter. This makes extracting and distributing the model for each PUF relatively easy because there are only two transistors per inverter that must be characterized.

It was shown that the Analog PUF makes a good PUF due to a relatively high uniqueness of 74%, and good reliability when using a strikingly low resolution of 3 bits to

balance the variance with temperature against the ability to differentiate PUFs. This design also took advantage of challenge engineering, increasing the performance of the PUF but adding a possible avenue of attack by simplifying the necessary Look Up Tables (LUTs).

It was also shown with preliminary testing of this design as a PPUF that model extraction, outlined in Figure 4 with results presented in Figure 5, of the push-pull amplifier is very promising [7].

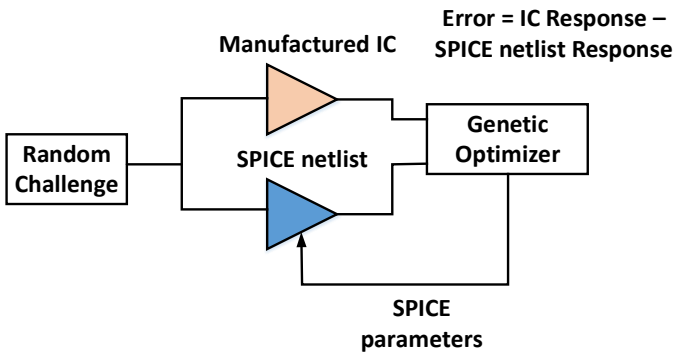


Figure 4 – PPUF modeling process

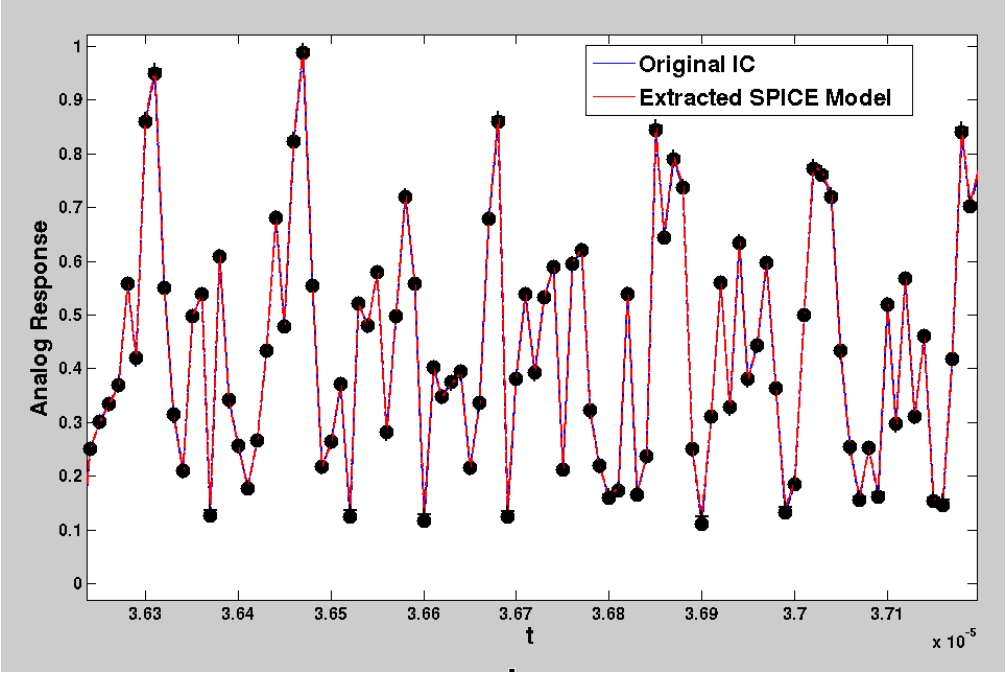


Figure 5 – Successful modeling of APPUF

The error in the output symbols was 0.38%, and it was shown that the error was primarily a result of quantization. Having this error as low as possible is important for a PPUF as the entire verification process is predicated on the accuracy of the given model.

3.2 Design Process

Most of the work toward this thesis was put into changing and iterating designs.

Of several alternate designs, the one most tested was a cascaded current mirror. The design was chosen due to the ability to place them in series, as seen in Figure 6, to make simulation harder. When carefully operated in the nonlinear region, these current mirrors showed promisingly large output variance with process variations. With so many transistors in series, however, modeling of a physical PPUF would be nearly impossible.

The design was eventually abandoned due to this modeling issue, as well as concerns over possible simulation shortcuts.

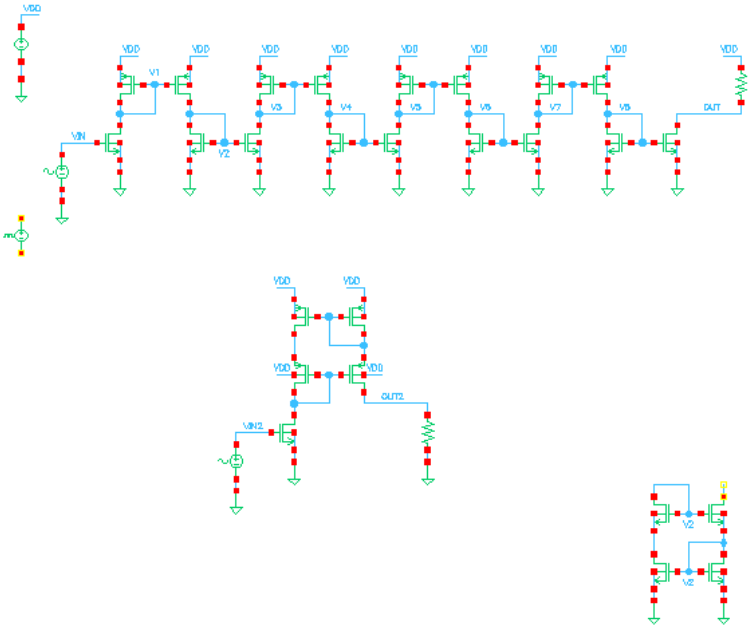


Figure 6 – Current mirror PPUF test bench

Modifications were also made to the push-pull amplifier design. The first and most obvious change was to move the difference amplification to before the conversion from analog to digital, as seen in Figure 7. This reduces the number of ADCs, and allows the ADC to capture the true range of the data of interest instead of the raw output of each push-pull amplifier.

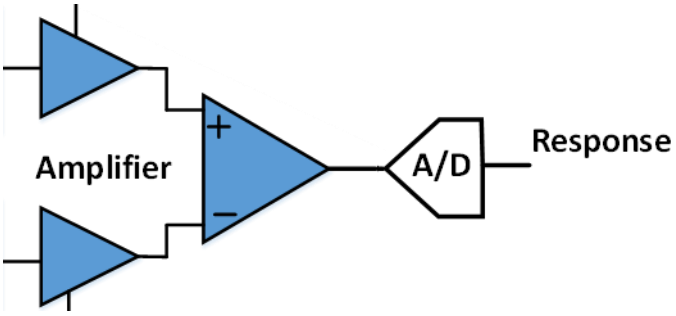


Figure 7 – New ADC and diff. amp placement

The next major change was to switch the input to a filtered bitstream instead of an engineered analog challenge. This allowed for higher simulation load to communication load ratios. The removal of challenge engineering reduces the risk of LUT-based attacks. The removal of the high-speed input DAC also considerably simplifies the APPUF and reduces the power consumption.

Next the bias selection seen in Figure 8 was added. Four levels were chosen for each power and ground. By creating a feedback between current draw at the output and power supply level, simulation was slowed down by a factor of four. Since the power supply changes once for every ten bits input to the push-pull amplifier, two distinct rates of charging and discharging are created, making periodic simulation shortcuts more difficult. Capacitors were added to the rail, carefully chosen to be large enough to give the supply a long memory but still small enough to stay on the proper timescale and allow discharging through the push-pull transistors to significantly change the voltage.

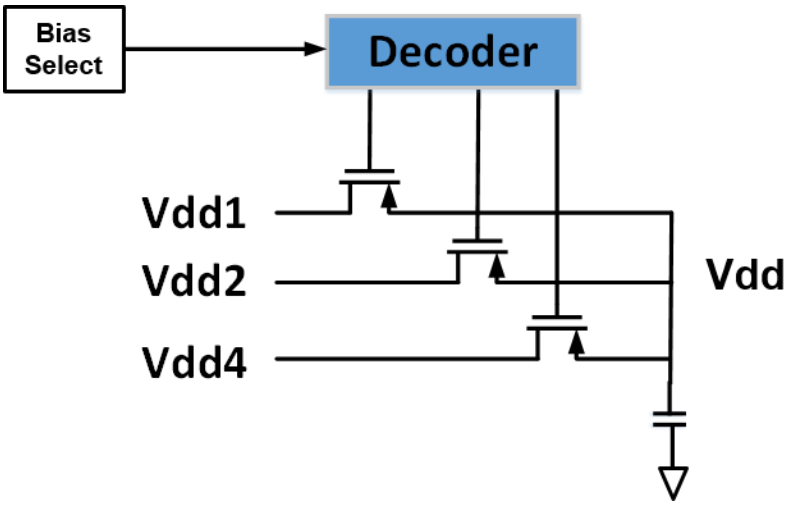


Figure 8 – Bias selection circuit (Vdd side)

Differentiating between APPUFs was difficult for very similar values between the two amplifiers. It was rare for all four Vth's to line up, but when it did happen the differential signal amplitude dipped into the tens of millivolts or smaller and the ADC had to be very high resolution to pick up the difference output. Since the amplitude of the difference was found to be constant with the same APPUF but different challenges, an analog normalizer was added before the ADC, as seen in Figure 9. An arbitrary preset challenge can be run through the APPUF, and the output scaled to fit the full scale range of the ADC. Subsequent challenges will then be observed at maximum resolution with a minimal loss of information due to scaling.

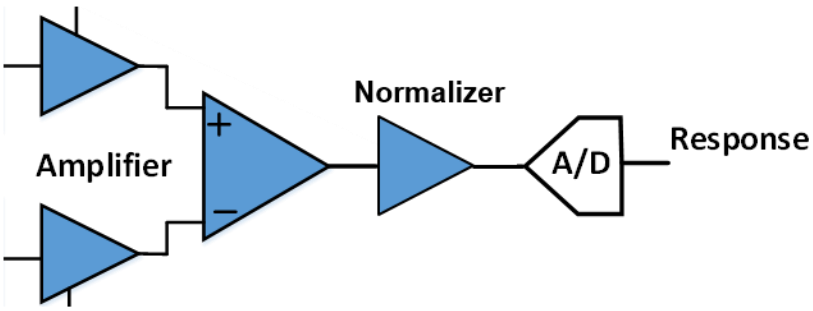


Figure 9 - Normalizer

Finally, results were varying widely over different temperatures. The circuit was designed to amplify process variations, but the same characteristics being observed also tend to change with temperature, as seen in Figure 13. This was solved by adding a temperature sensor. Models could be found for the APPUF over a range of temperatures, then when an APPUF is to be verified its temperature is read and the correct model for that temperature can be simulated to check against with minimal variance. The filtering was also modified so that temperature changes could not attenuate the signals enough to cause a loss of process variable information.

3.3 Final Design

The APPUF takes a bitstream challenge of any length from the verifier. For every fourteen bits, ten are fed identically into the amplifiers, and two are fed into each of the rail selects for the amplifiers. Each amplifier has an LPF to smooth the bitstream input which is put through the push-pull amplifiers with resistive biasing and HPF capacitors. The two-bit words applied to the bias selection are used to choose between four power rail levels and four ground levels. Each amplifier has nominally identical but separate circuits for this,

sharing only the input word and supplies. A capacitor smooths each the rail voltages and allows for charging and discharging through the output.

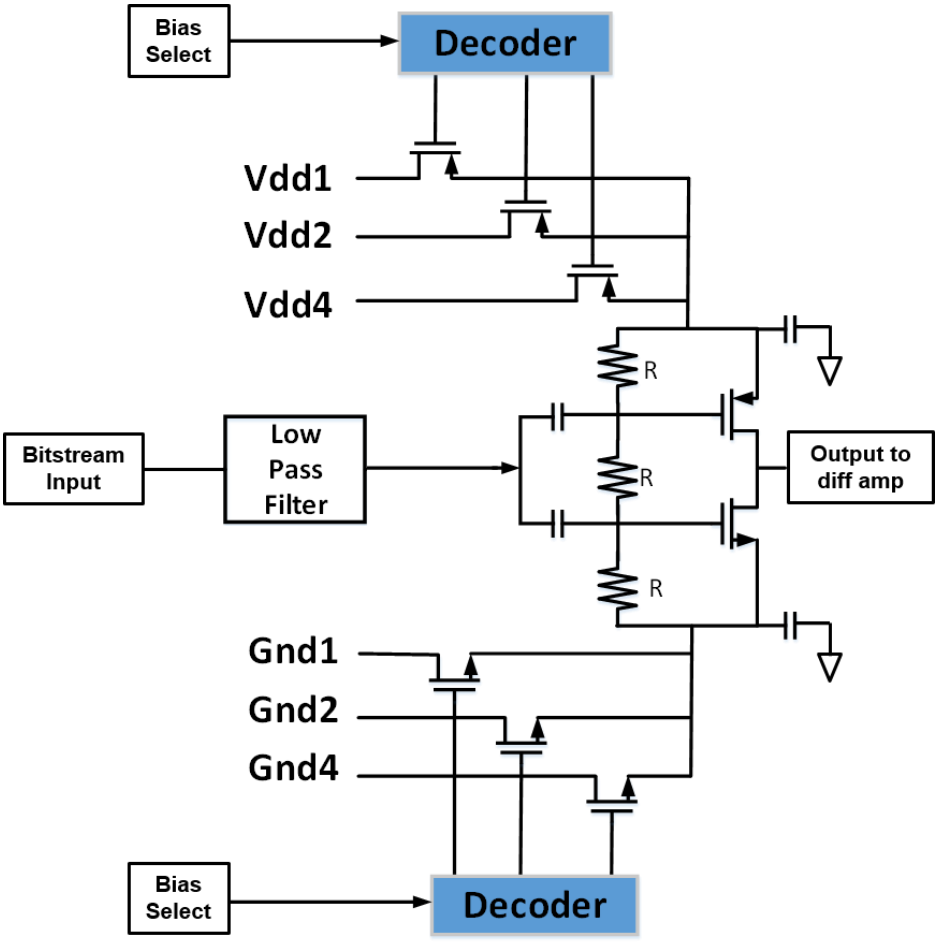


Figure 10 – Push-pull amplifier with filtering and bias select

The difference between the amplifier outputs is then taken. Since both nominally identical amplifiers are receiving the same challenge and bias, the only difference at the output will be as a result of the process variations present in the circuits. The difference signal is normalized to the full scale of the ADC, then converted to a three-bit output response to be sent back to the verifier.

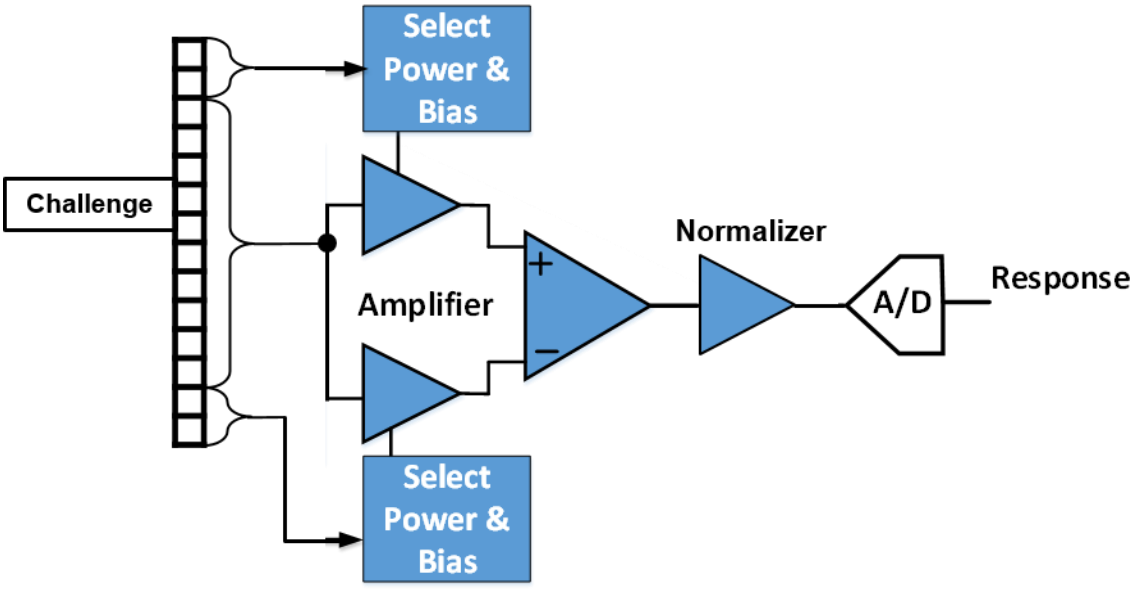


Figure 11 – APPUF diagram

CHAPTER 4. ANALOG PPUF ANALYSIS

4.1 Analysis Process

Design and simulation of the APPUF was performed in Cadence's Virtuoso tool, with simulation using the Spectre simulator. Batches of simulations were originally run via Ocean scripts for the first several months, but this quickly became tedious for the number of small changes that had to be made to the design and simulation environments. Since Matlab was used as a shell for setting up and starting simulations, it was much more efficient to interface directly with the Analog Design Environment GUI for importing settings and kicking off simulation runs.

Matlab did most of the heavy lifting for this project, automatically generating stimulus files, process corners and much more. Since storage of output data quickly became an issue, only the single analog output between the differential amplifier and the normalizer was recorded in Cadence's ".psf" file format. Normalization and quantization were performed in Matlab to streamline the process. See Appendix B for more details.

Unless otherwise noted, all simulations shown below were recorded using a sample of 200 PPUFs, run for a length of 5us execution time with the output ADC taking 3000 samples of 3 bits each, at temperatures of -20°C, 27°C and 120°C.

4.2 Results

In Figure 12 we can see the input challenge bitstream, the outputs of each push-pull amplifier, and the normalized difference taken between the two, which is the input to the ADC. The amplifiers function as poor inverters, with enough variance in their shape to make the difference output hard to predict.

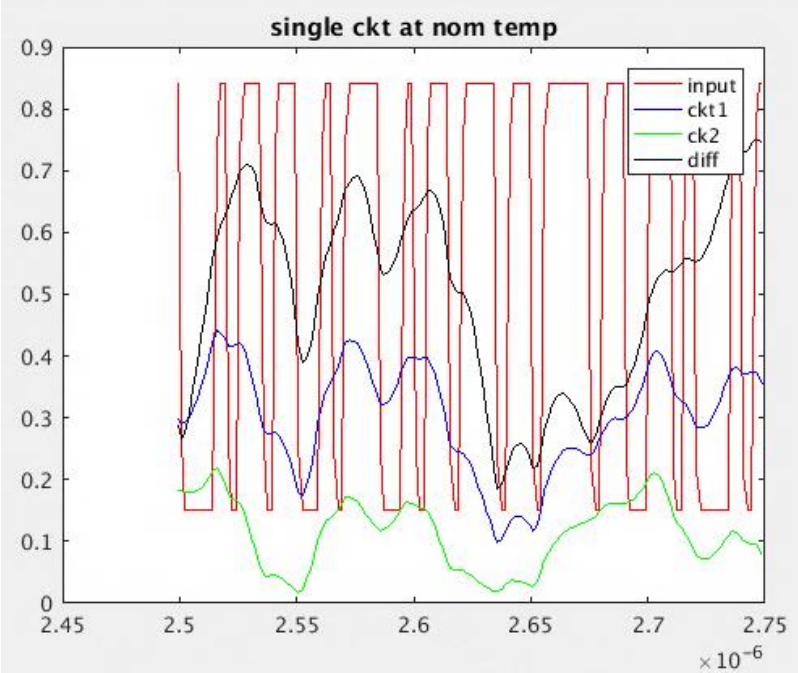


Figure 12 – Signals from random sample APPUF: bitstream, output of each push-pull amplifier, and the differential input to the ADC

One of the important metrics we identified was reliability over a range of temperatures. Since the APPUF is necessarily sensitive to temperature as covered above, the most important metric is that APPUF outputs still retain a distinctive shape so that they can be differentiated from one another at each temperature, as seen in Figure 13.

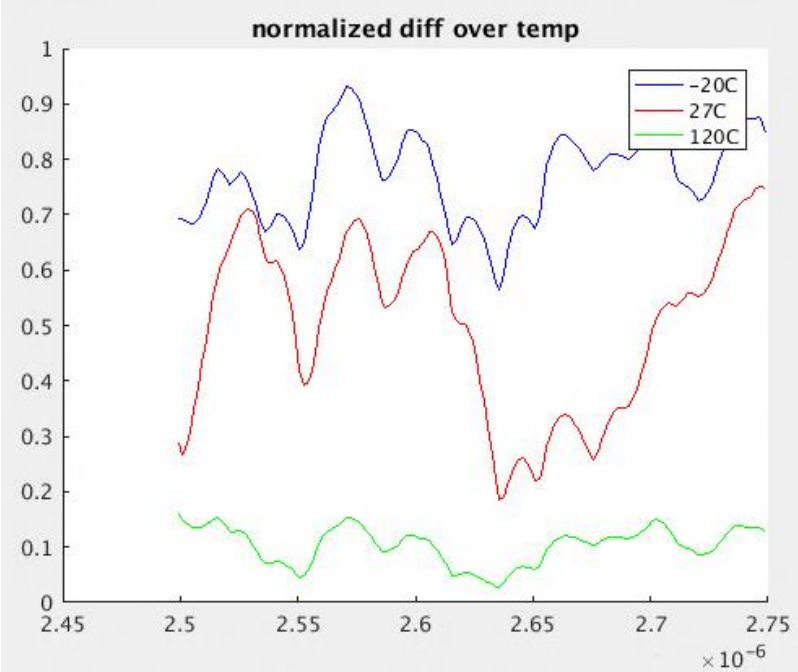


Figure 13 – Input to the ADC for the same APPUF over temperature range

Temperature variability is further analyzed when we start looking at pairwise distance metrics below.

Next we start analysis of a batch of 200 APPUFs with normally distributed process variation over ±20% Vth, σ = 6%. First, we can see that the inputs to the ADC are well distributed over the full-scale range in Figure 14.

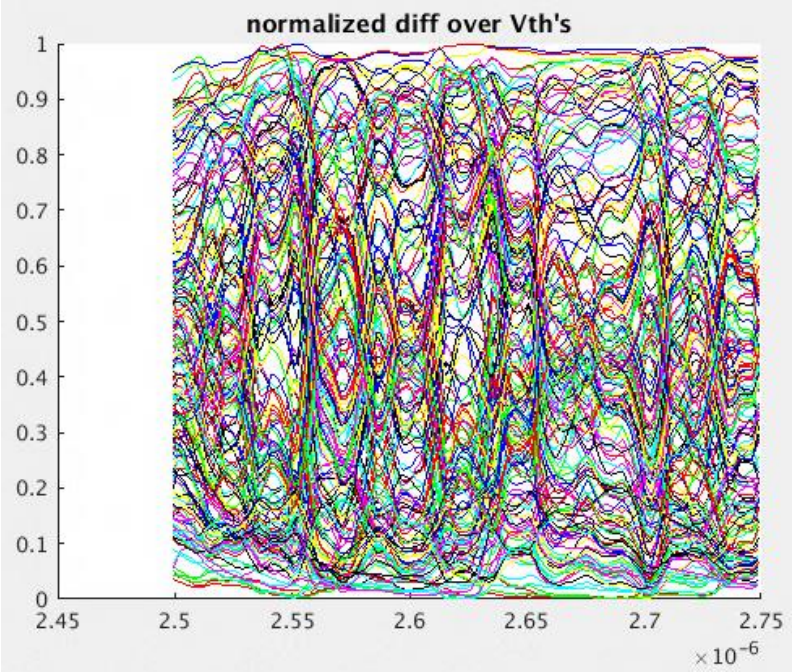


Figure 14 – Input to the ADC for 200 different APPUFs

The two most similar inputs to the ADC of the 200 APPUFs by two different distance metrics, Euclidian and Manhattan distance, are shown in Figure 15.

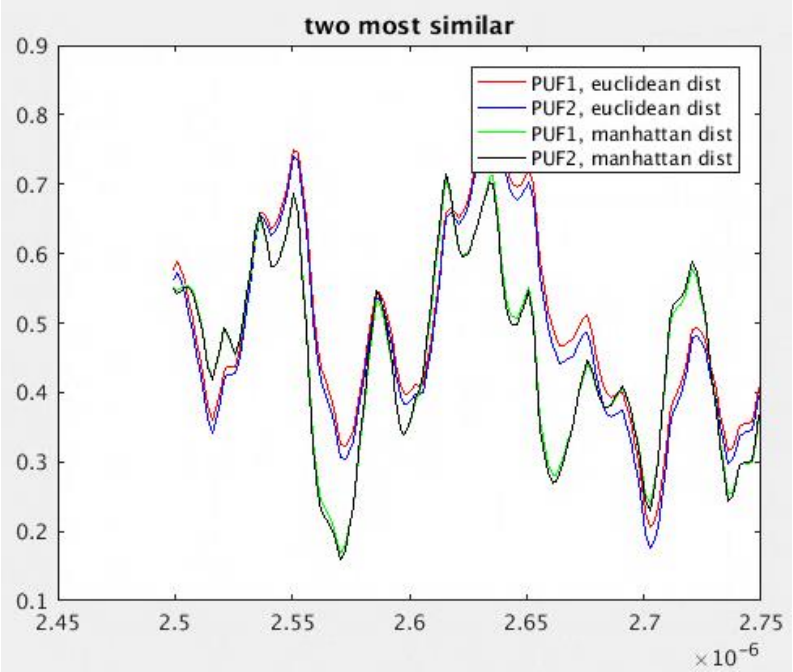


Figure 15 – Input to the ADC for most similar pair of APPUFs for two different measurement methods

This supports the use of Manhattan distance as a measure of the "closeness" of two APPUFs, while showing that the human eye can still differentiate them, if only barely.

Passing this through the ADC, we get the final output for verification in Figure 16.

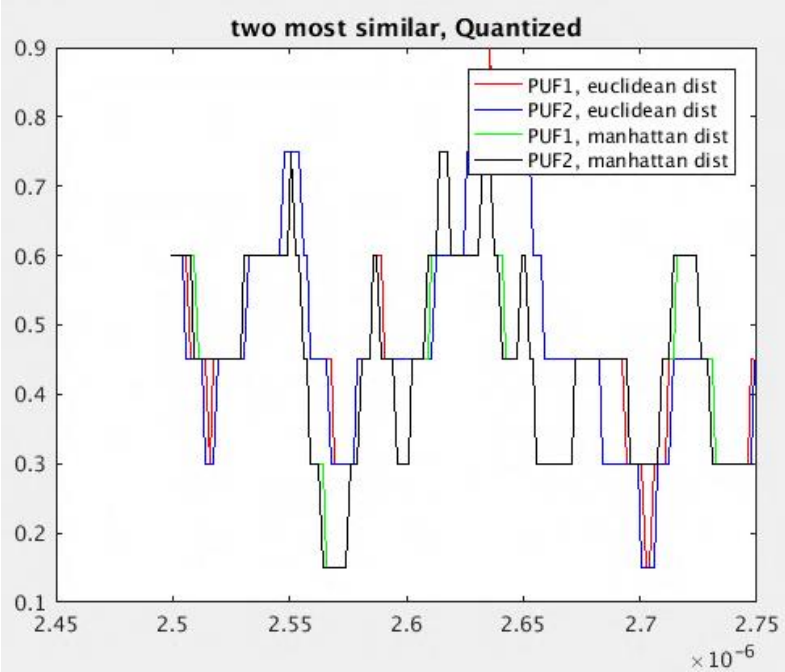


Figure 16 – Output of the ADC for most similar pair of APPUFs for two different measurement methods

Again we can see that with only 3 bits per sample, we can differentiate even the most similar of APPUFs. Assuming a perfectly accurate model can be created, we can identify and differentiate every one of the 200 APPUFs at all temperatures from -20°C to 120°C.

To dive further into differentiability, we next look at the distance between pairs of APPUFs. The traditional method of distance between PUFs is Hamming distance, with an average Hamming distance of 50% indicating maximum entropy. In the histogram of Figure 17 we can see that the minimum Hamming distance was 4.59%, with the worst average being 49.43% across minimum, nominal and maximum temperatures.

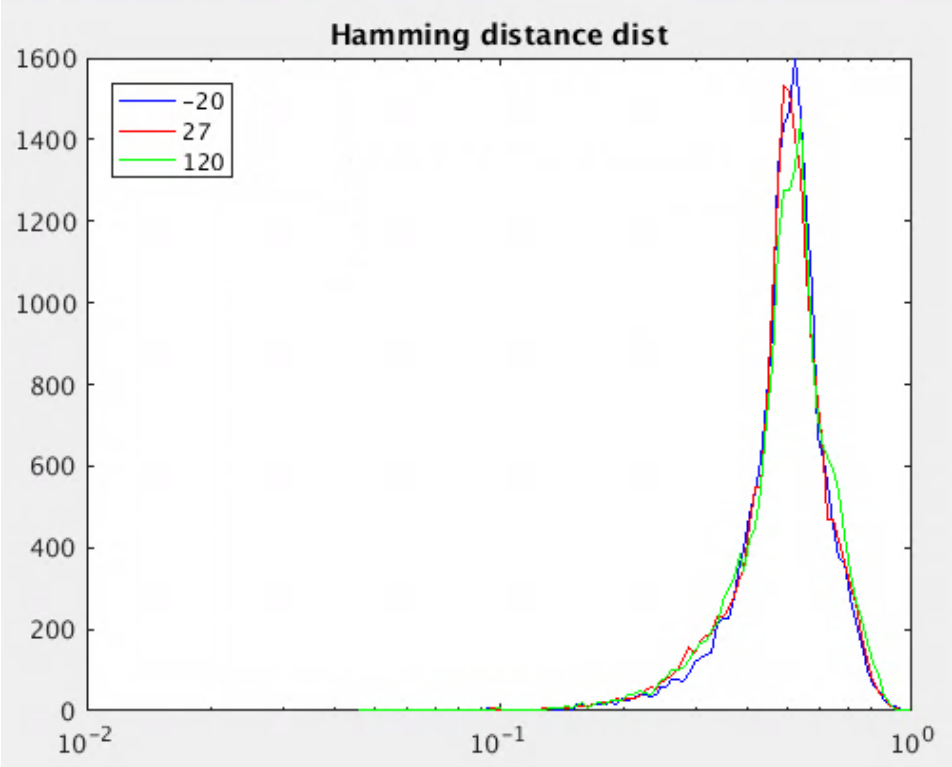


Figure 17 – Histogram distribution of pairwise Hamming distances for 200 process varied APPUFs across range of temperatures

This is not a proper metric for our APPUF, however, due to the analog nature of our true output. A better distance metric is the Manhattan distance, whose distribution can be seen in Figure 18.

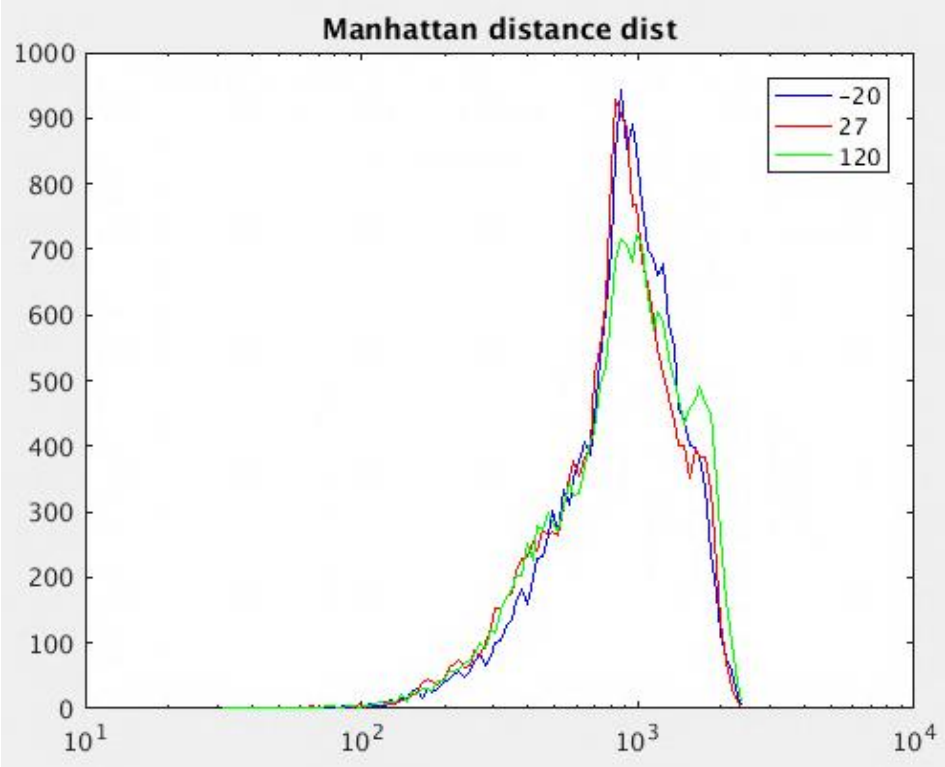


Figure 18 – Distribution of pairwise Manhattan distances for 200 APPUFs taken before the ADC

Manhattan distance more accurately captures the arithmetic distance between two analog signals, without amplifying noisy outliers like the Euclidean measures used in previous work by Deyati et al. do.

Furthermore, the Manhattan distance is well preserved through the quantization process, even with only 3 bits as seen in Figure 19.

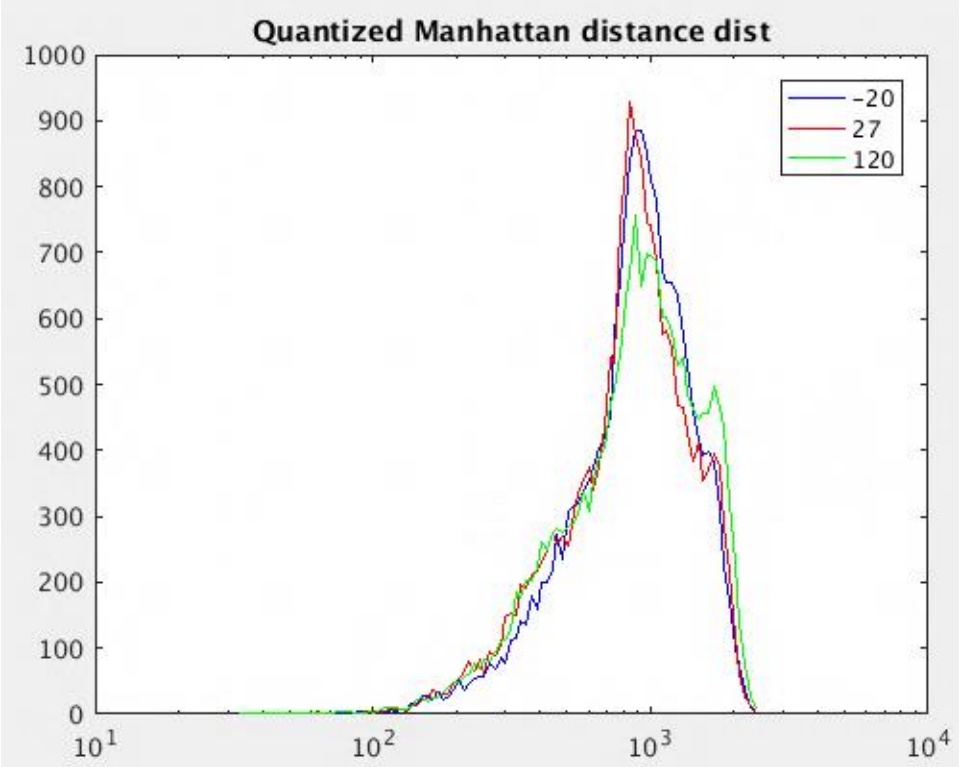


Figure 19 – Distribution of pairwise Manhattan distances for 200 APPUFs taken after the ADC

Using the Manhattan distance between the APPUF being verified and our APPUF model to determine identity gives us considerable leeway in choosing where to cut off clustering radii depending on the desired yield and noise floor.

Deyati et al. already showed the feasibility of highly accurate modeling of his Analog PUF [7]. Since the heart of inverter is the same, it follows that modeling will be just as successful with the new design. Unfortunately, creating a production-worthy model builder would take considerable expertise and months of time, and drawing conclusions from a sub-standard model builder would be meaningless. Instead, we can look at the behavior of our APPUF in Figure 20 and Figure 21 for clues.

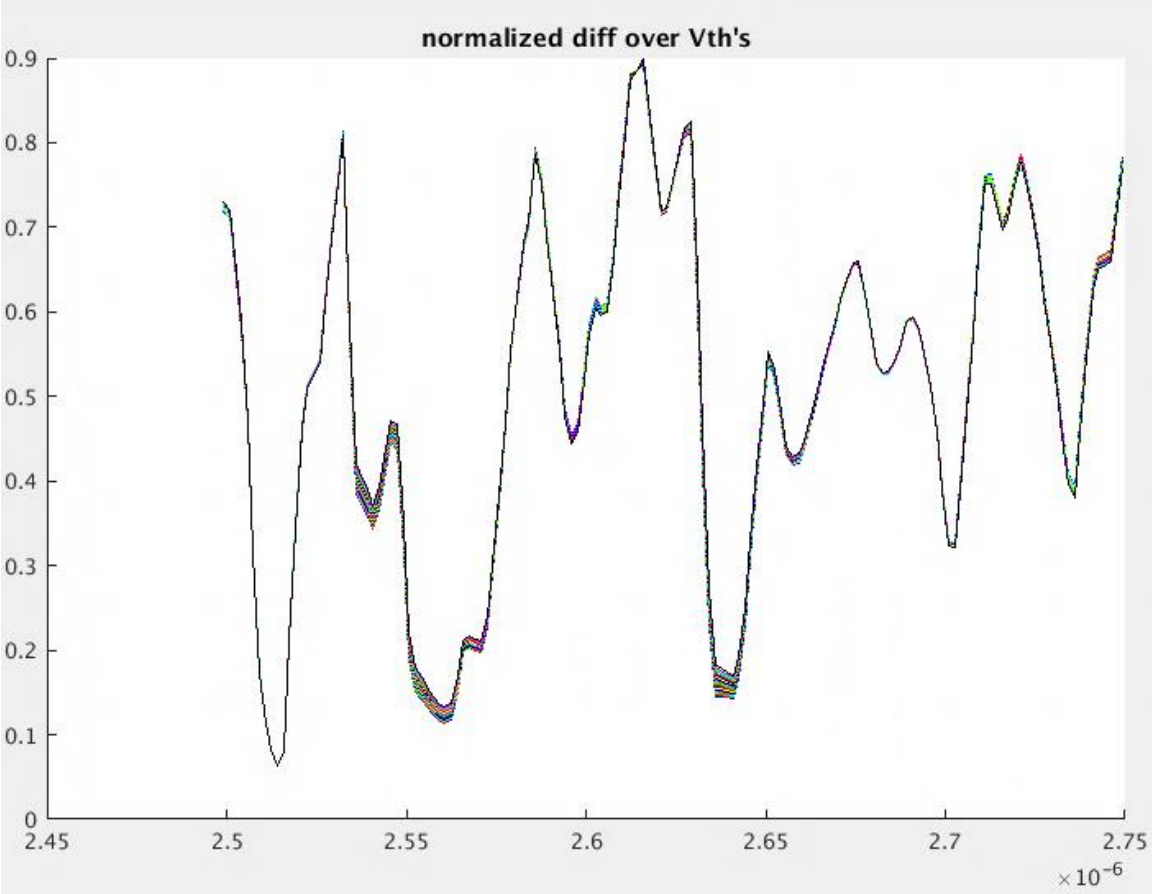


Figure 20 – Input to the ADC for ±1% variance of a single Vth

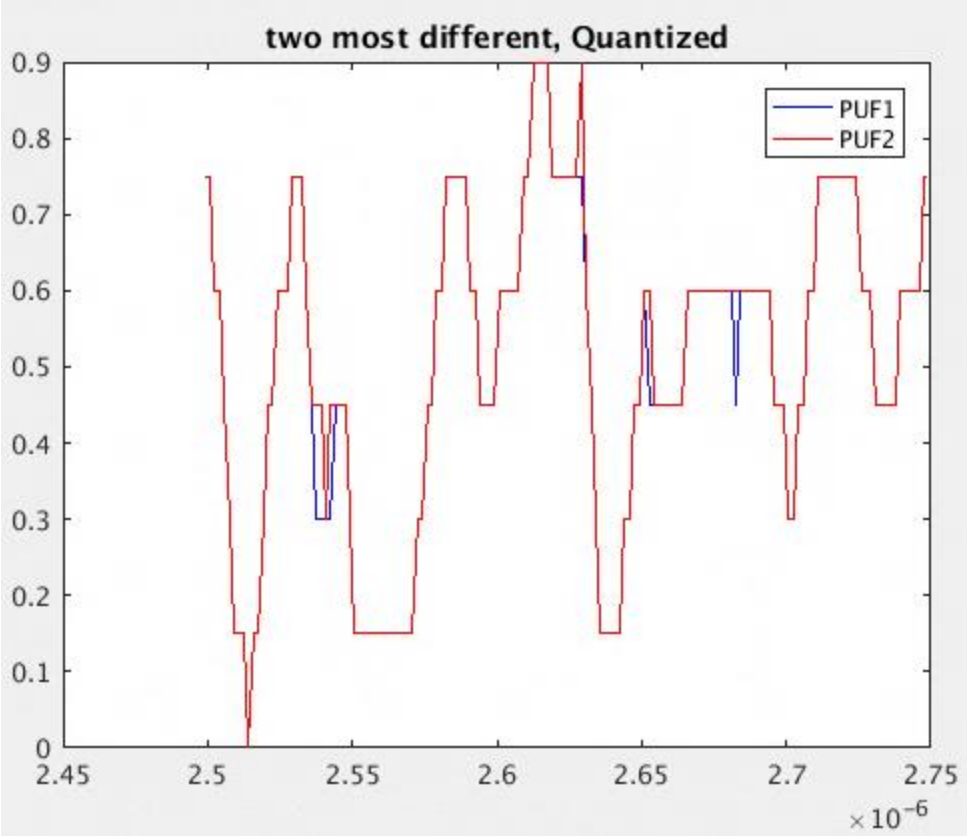


Figure 21 – ADC output for two APPUFs identical except for 2% change in a single Vth value

Although Figure 14 shows that the waveform shape of the APPUF output varies widely and in a hard-to-predict manner between APPUFs, Figure 20 shows us that with small, predictable changes of circuit parameters we see similarly small, predictable changes in the output. This not only bodes well for model building, but also suggests that APPUF models can remain accurate with device aging and other nonidealities, as seen in Figure 21. If component drift can be predicted, the output models can be adjusted with no potentially costly and vulnerable recalibrations necessary.

Finally, the APPUFs were tested for simulation time. Running Cadence's Spectre circuit simulator on the fastest settings available, an ESG ratio of about 231,165 was achieved. This is linear over increasing challenge lengths, as shown in Figure 22.

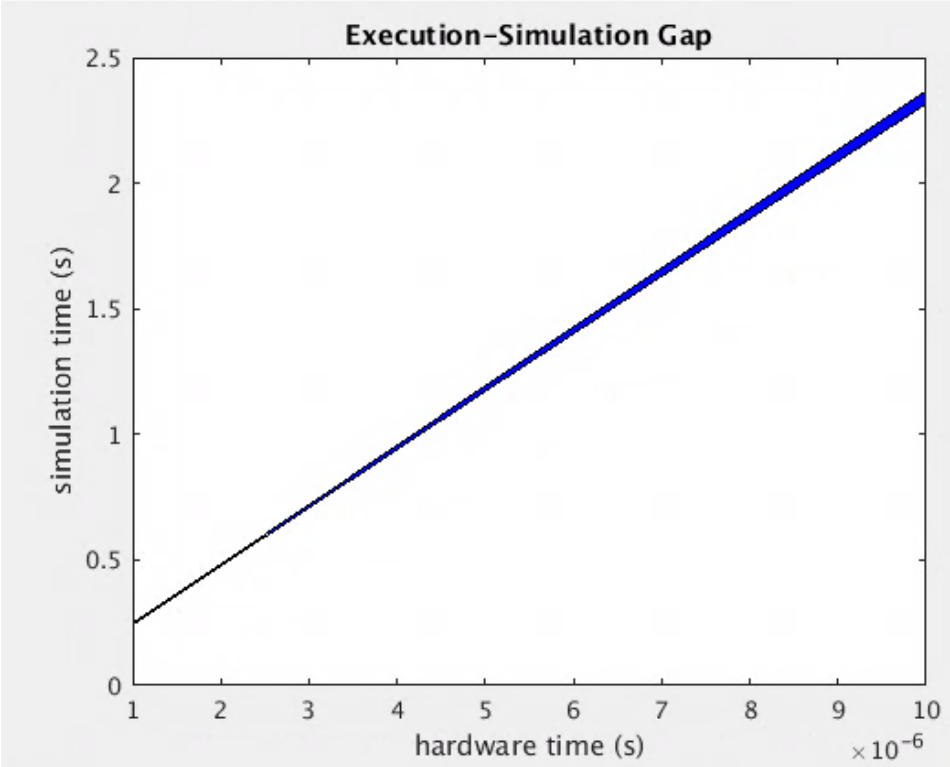


Figure 22 – ESG scaling

The width of the line shows the slight variability from run to run in how fast Spectre can solve the circuit.

CHAPTER 5. CONCLUSION

This report proposes a modified design for a new Analog Public Physically Unclonable Function and analyzes its fitness for use as such. The APPUF is topically salient in a world increasingly concerned with hardware vulnerability to cyber-attacks [8].

5.1 Fitness Metrics Revisited

In order to show that our design is useful as a PPUF, we have observed indicators of each of the stated metrics for a PPUF. It was shown that the APPUF has 100% uniqueness for 200 random samples. The APPUF was reliable over temperature, maintaining its uniqueness even at extreme temperatures of -20°C and 120°C. We have shown the APPUF to have a six figure ESG, making simulation attacks impossible when verification is performed properly. With a carefully designed circuit, we have ensured that although the ESG could be reduced further with parallelization and clever simulation shortcuts, there are enough feedback loops and nonlinearities that the ESG would still be orders of magnitude higher than necessary. Since the execution time of our APPUF is inherently scalable, the ESG can be further exploited to overcome slow communications that could otherwise mask long simulation times. The APPUF is also intrinsically resistant to reverse engineering attacks because it is a deceptively simple nonlinear circuit. More complex circuits, such as an Arbiter PUF or even a NanoPPUF, are much easier to engineer because their relatively simple building blocks can be reproduced via other methods and assembled to mimic the whole. Finally, the one metric not addressed previously is entropy in an APPUF. One might posit that entropy is not nearly as relevant to an analog system as

it is to an Arbiter PUF, in which single bits are given as output. As stated in the introduction, entropy is really an indicator of uniqueness and resistance to attack; since it can be shown that we have both, a direct measure of entropy becomes less necessary. Still, Figure 14 in conjunction with our average Hamming distance of practically 50% provide a basis for stating that the APPUF design has adequate entropy to be useful as a PPUF.

5.2 Future Work

The obvious next step would be to get a batch of APPUFs laid out and manufactured so that they can be modeled and tested against the simulated results here. Before manufacturing, however, there are a few more areas to be exhausted in the name of rigor.

First, a definite model building process should be laid out and tested with the final APPUF design. Next, this model should be tested against a large sample of realistically varied APPUFs to get a score of uniqueness more accurate to a physically implemented PPUF. This can inform decisions about clustering beyond the current method of giving each APPUF its own cluster because they are all distinct.

Next, a more thorough examination of simulation techniques, model defeat and entropy should be conducted. Ideally, an expert third party could be recruited to work on attacking the APPUF.

APPENDIX A. DESIGN VALUES

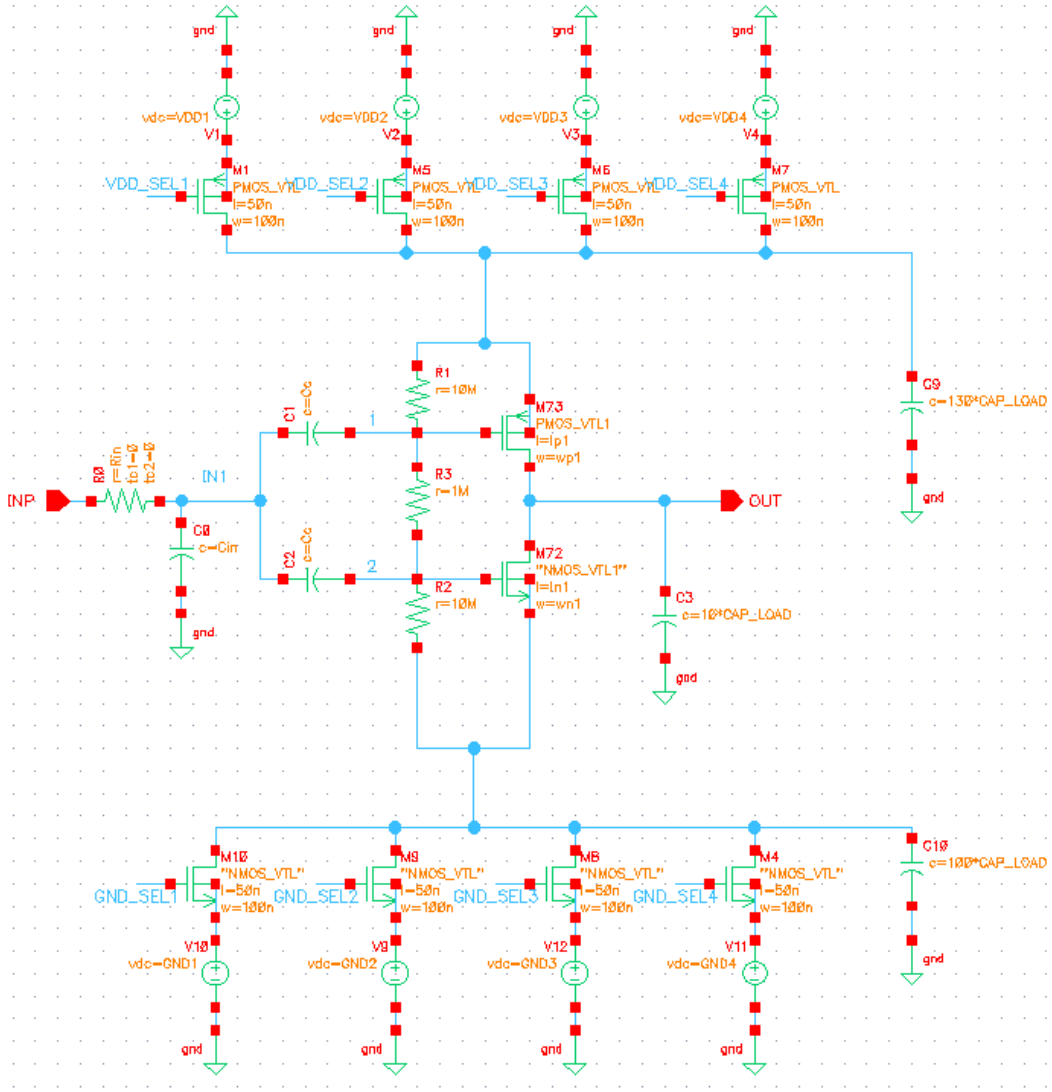


Figure 23 – Virtuoso Schematic with labels

Table 1 – Final Design Values

Amplifier Design Values		
Variable	Value	Description
GND1	0V	1st ground select
GND2	50mV	2nd ground select
GND3	100mV	3rd ground select
GND4	150mV	4th ground select
VDD1	1.2V	1st power select
VDD2	1.08V	2nd power select
VDD3	960mV	3rd power select
VDD4	840mV	4th power select
CAP_LOAD	10fF	used for filtering outputs
Cc	1fF	Input HPF
Cin	2pF	Input LPF
Rin	2.5kΩ	Input LPF
Ln1	50nS	Length of push-pull NMOS
Lp1	50nS	Length of push-pull PMOS
Wn1	50nS	Width of push-pull NMOS
Wp1	150nS	Width of push-pull PMOS
Vtn_nom	471mV	Nominal threshold voltage NMOS
Vtp_nom	-423mV	Nominal threshold voltage PMOS

APPENDIX B. MATLAB SNIPPETS

```
sim=33;
numTemp=3;
% numV=length(Vtn1);
numV=200;
simTime=5E-6;
simSteps=3000;
VDD=1.2;
tinterp=linspace(0,simTime,simSteps)';
colors=['b','r','g','c','m','k','y'];
while length(colors)<numV
    colors=[colors colors];
end
clear sigi sigo sigo1 sigo2;

for temp=1:numTemp
    for v=1:numV
        sigi(temp,v)=cds_srr(sprintf(['/nethome/kgassel3/simulation/PUF_Saby/PUSHPULL_PPUFkyle/adexl_corners/results/'...
            'data/Interactive.%d/%d/PUF_Saby:PUSHPULL_PPUFkyle:1/psf'],sim,v+(temp-1)*numV), 'tran-tran', 'INP');
        sigo(temp,v)=cds_srr(sprintf(['/nethome/kgassel3/simulation/PUF_Saby/PUSHPULL_PPUFkyle/adexl_corners/results/'...
            'data/Interactive.%d/%d/PUF_Saby:PUSHPULL_PPUFkyle:1/psf'],sim,v+(temp-1)*numV), 'tran-tran', 'OUT');
        sigo1(temp,v)=cds_srr(sprintf(['/nethome/kgassel3/simulation/PUF_Saby/PUSHPULL_PPUFkyle/adexl_corners/results/'...
            'data/Interactive.%d/%d/PUF_Saby:PUSHPULL_PPUFkyle:1/psf'],sim,v+(temp-1)*numV), 'tran-tran', 'OUT0');
        sigo2(temp,v)=cds_srr(sprintf(['/nethome/kgassel3/simulation/PUF_Saby/PUSHPULL_PPUFkyle/adexl_corners/results/'...
            'data/Interactive.%d/%d/PUF_Saby:PUSHPULL_PPUFkyle:1/psf'],sim,v+(temp-1)*numV), 'tran-tran', 'OUT1');
        sigiterp(:,temp,v)=interp1(sigi(temp,v).time,sigi(temp,v).V,tinterp);
        sigiterp(end,temp,v)=sigi(temp,v).V(end);
        sigoterp(:,temp,v)=interp1(sigo(temp,v).time,sigo(temp,v).V,tinterp);
        sigoterp(end,temp,v)=sigo(temp,v).V(end);
        sigo1terp(:,temp,v)=interp1(sigo1(temp,v).time,sigo1(temp,v).V,tinterp);
        sigo1terp(end,temp,v)=sigo1(temp,v).V(end);
        sigo2terp(:,temp,v)=interp1(sigo2(temp,v).time,sigo2(temp,v).V,tinterp);
        sigo2terp(end,temp,v)=sigo2(temp,v).V(end);
    end
end
len=zeros(numTemp,numV);
for k=1:numV
    for i=1:numTemp
            len(i,k)=length(sigo(i,k).V);
    end
end
lenMin=min(min(len))
lenMax=max(max(len))
clear sigi sigo sigo1 sigo2;
sigi=sigiterp;
sigo=sigoterp;
sigo1=sigo1terp;
sigo2=sigo2terp;
clear sigiterp sigoterp sigo1terp sigo2terp;

for i=1:numTemp
    sigomax=max(sigo(round(end/10):end,:,:));
    sigomin=min(sigo(round(end/10):end,:,:));
    sigoNorm=(sigo-sigomin)./(sigomax-sigomin);
end

sigo=sigoNorm;
```

Figure 24 – Data import, sampling and normalization

```
bits=3;
sigoD=round(sigo./VDD.*2^bits);
sigoD(sigoD(:)<0)=0;
sigoD(sigoD(:)>=2^bits)=2^bits-1;
sigoQ=sigoD.*VDD./2^bits;
sigoDbin=zeros(simSteps*bits,numTemp,numV);
sigoDbin(:)=(dec2bin(sigoD(:),bits)-'0').';
```

Figure 25 – Quantization

```
for k=1:numTemp
    count=0;
    for i=1:numV
        for j=i+1:numV
            count=count+1;
            arithV(k,count)=sum((sigo(:,k,i)-sigo(:,k,j)).^2);
            pairV(1:2,count)=[i,j];
            manhtnV(k,count)=sum(abs(sigo(:,k,i)-sigo(:,k,j)));
            manhtnQ(k,count)=sum(abs(sigoQ(:,k,i)-sigoQ(:,k,j)));
            maxDiff(k,count)=max(abs(sigo(:,k,i)-sigo(:,k,j)));
            quantSame(k,count)=min(sigoQ(end/10:end,k,i)==sigoQ(end/10:end,k,j));
            hamV(k,count)=pdist2(sigoDbin(:,k,i).',sigoDbin(:,k,j).','hamming');
        end
    end
end
fprintf('number of pairs indistinguishable: %d\n',max(max(quantSame)));
```

Figure 26 – Distance measures

```
tempL=-20;
tempN=27;
tempH=120;
numPUF=20;
Vtn1=zeros(1,numPUF);
Vtp1=zeros(1,numPUF);
Vtn2=zeros(1,numPUF);
Vtp2=zeros(1,numPUF);
for i=1:numPUF
    while 0.3768>=Vtn1(i) || Vtn1(i)>=0.5652 || -0.5076>=Vtp1(i) || Vtp1(i)>=-0.3384 ||...
            0.3768>=Vtn2(i) || Vtn2(i)>=0.5652 || -0.5076>=Vtp2(i) || Vtp2(i)>=-0.3384
        Vtn1(i)=normrnd(0.471,0.028);
        Vtp1(i)=normrnd(-0.423,0.028);
        Vtn2(i)=normrnd(0.471,0.028);
        Vtp2(i)=normrnd(-0.423,0.028);
    end
end

save VthVals Vtn1 Vtp1 Vtn2 Vtp2;
```

Figure 27 – Threshold voltage generation

```
fp=fopen('setupTry.sdb','w');

fprintf(fp,'<?xml version="1.0"?> \n<setupdb version="5">setuptmp \n    <active>Active Setup \n');
fprintf(fp,'        <tests>\n           <test enabled="1">PUF_Saby:PUSHPULL_PPUFkyle:1 \n');
fprintf(fp,'                <tool>ADE</tool>\n          </test>\n       </tests>\n');
fprintf(fp,'        <corners>\n         <corner enabled="0">_default</corner>\n');

for temp=[tempL,tempN,tempH]
    for i=1:numPUF
        fprintf(fp,'            <corner enabled="1">v%.0ft%.0f \n',i,temp);
        fprintf(fp,'                <vars>\n                    <var>temperature\n');
        fprintf(fp,'                        <value>%.0f</value>\n',temp);
        fprintf(fp,'                    </var>\n                    <var>nmos_vtl_saby1\n');
        fprintf(fp,'                        <value>%f</value>\n',Vtn1(i));
        fprintf(fp,'                    </var>\n                    <var>pmos_vtl_saby1\n');
        fprintf(fp,'                        <value>%f</value>\n',Vtp1(i));
        fprintf(fp,'                    </var>\n                    <var>nmos_vtl_saby2\n');
        fprintf(fp,'                        <value>%f</value>\n',Vtn2(i));
        fprintf(fp,'                    </var>\n                    <var>pmos_vtl_saby2\n');
        fprintf(fp,'                        <value>%f</value>\n',Vtp2(i));
        fprintf(fp,'                    </var>\n                </vars>\n           </corner>\n');
    end
end
fprintf(fp,'        </corners>\n    </active>\n <history>History</history>\n</setupdb>');

fclose(fp);
```

Figure 28 – Writing corners to file for import

```
Range =[0.15 0.84];
RangeRail=[0.15 0.84];
%Range =[0 1.2];
N=10000;
tstep=5e-9;
t=0:tstep:(N-1)*tstep;
Vin=randi([0 1],1,N).*(Range(2)-Range(1))+Range(1);


factor=10;
Nrail=N/factor;
tstepRail=tstep*factor;
tRail=0:tstepRail:(Nrail-1)*tstepRail;

VDD_SEL=randi([1 4],1,Nrail);
GND_SEL=randi([1 4],1,Nrail);

VDD_SELbin=~[VDD_SEL==1;VDD_SEL==2;VDD_SEL==3;VDD_SEL==4];
GND_SELbin=[GND_SEL==1;GND_SEL==2;GND_SEL==3;GND_SEL==4];
```

```
%%

WriteStimBit(file1,'w','INP',Range(1),Range(2),tstep/5,tstep,randi([0 1],1,N));
WriteStimBit(file2,'w','INP',Range(1),Range(2),tstep/5,tstep,randi([0 1],1,N));
for i=1:4
    WriteStimBit(file2,'a',strcat('VDD_SEL',num2str(i)),RangeRail(1),RangeRail(2),1e-9,tstepRail,VDD_SELbin(i,:));
    WriteStimBit(file2,'a',strcat('GND_SEL',num2str(i)),RangeRail(1),RangeRail(2),1e-9,tstepRail,GND_SELbin(i,:));
end
```

Figure 29 – Create challenge bitstream

```
function[]=WriteStimBit(stimulusfile,mode,input_node,val0,val1,trf,period,data)

fp=fopen(stimulusfile,mode);
if strcmp(mode,'w')
    fprintf(fp,'simulator lang=spectre \n');
end

vsource=strcat('V',num2str(input_node));
vsource=strcat(vsource,{' '},{'( '},num2str(input_node),{' '},{' 0 ) '});
fprintf(fp,'\n\n%s',vsource{:});
fprintf(fp,'  vsource type=bit val0=%f val1=%f rise=%.0fe-12', ...
    val0,val1,trf*(1e12));
fprintf(fp,' fall=%.0fe-12 period=%.0fe-12 data="%s"\n\n', ...
    trf*(1e12), period*(1e12), sprintf('%d',data));

fclose(fp);
end
```

Figure 30 – Write stimulus to file

REFERENCES

[1] Q. Guo, J. Ye, Y. Gong, Y. Hu and X. Li, "Efficient Attack on Non-linear Current Mirror PUF with Genetic Algorithm," *2016 IEEE 25th Asian Test Symposium (ATS)*, pp. 49-54, 2016.

[2] M. Potkonjak and V. Goudar, "Public physical unclonable functions," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1142--1156, 2014.

[3] N. Beckmann and M. Potkonjak, "Hardware-based public-key cryptography with public physically unclonable functions," *Proc. Int. Workshop Inf. Hiding*, p. 206–220, 2009.

[4] S. Wei, J. B. Wendt, A. Nahapetian and M. Potkonjak, "Reverse engineering and prevention techniques for physical unclonable functions using side channels," *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1-6, 2014.

[5] J. B. Wendt and M. Potkonjak, "The bidirectional polyomino partitioned PPUF as a hardware security primitive," *2013 IEEE Global Conference on Signal and Information Processing*, pp. 257-260, 2013.

[6] M. Li, J. Miao, K. Zhong and D. Z. Pan, "Practical public PUF enabled by solving max-flow problem on chip," *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1-6, 2016.

[7] S. Deyati, B. J. Muldrey, A. D. Singh and A. Chatterjee, "Challenge Engineering and Design of Analog Push Pull Amplifier Based Physically Unclonable Function for Hardware Security," *2015 IEEE 24th Asian Test Symposium (ATS)*, pp. 127-132, 2015.

[8] M. Benantar, "The Internet public key infrastructure," *IBM Systems Journal*, vol. 40, no. 3, pp. 648-665, 2001.